JEDx

U.S. Chamber of Commerce Foundation

API DESIGN

Application programming interface to support public-private approaches for enhanced collection and use of standards-based jobs and employment data.

AUGUST 15, 2022

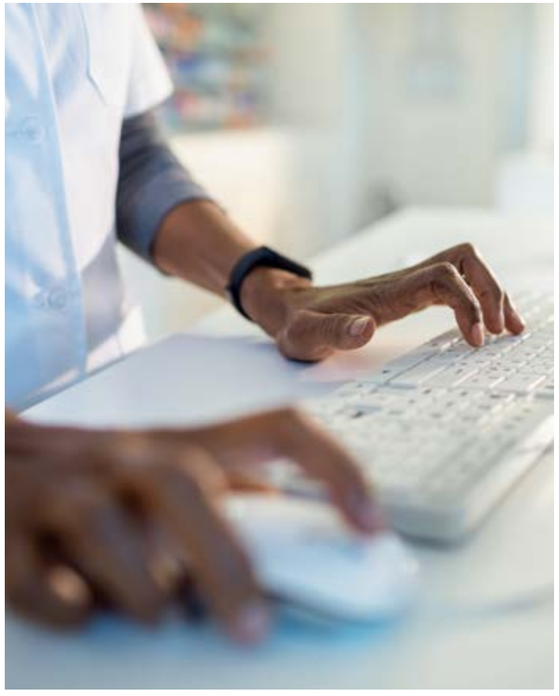
CONTENTS

This public-private approach has the potential to substantially reduce reporting costs for employers while also improving data quality. What's more, JEDx will transform our understanding of the labor market and empower workers with data.

—JASON A. TYSZKO

EXECUTIVE SUMMARY & OVERVIEW

OVERVIEW[1]



The technical application-programming-interface (API)-focused extension of the Architectural Options Working Document is a living document, a resource for researching and developing high-level architectural options for standards-based data collection and exchange of jobs and employment data (JEDx).

Feedback from multiple stakeholders informed this design, which is intended to support JEDx pilot programs while also serving the long-term JEDx vision. The goal of this API is to be a transport layer for the JDX+ standard data definitions and model maintained by the HR Open Standards Consortium (HR Open Standards) and proposed JEDx updates to those standards. The specification also relies on work done on The Open Application Group Inc.'s (OAGi's) Representational State Transfer or Representational Entity State Transfer (RESTful) Web API Design Version 2.0[3] and uses the Standardized Infrastructure open standards supported by several standards bodies, including the Postsecondary Education Standards Council (PESC), Access For Learning (A4L),[6] Medbiquitous, and HR Open Standards Consortium.[8]

[1] Eudy K., Overview Scope and Related Scope taken from Architectural Options Working Document. Retrieved April 21, 2022, from https://docs.google.com/document/d/1HXLqCGWHglpVv_SuzHdOTypOsE9uGY8K/edit

[2] Eudy, 2022

[3] Fohn S., OAGi_RESTful_Web_API_Design_EN-US. August 2019.

[4] Lovell J., DSU Standardized Infrastructure. May 2020. https://docs.google.com/document/d/1WrN_iCfrFPYHkVk3nxwj5hbQsgezbVEXI05WH3oKUP4/edit?usp=sharing

[5] https://pesc.org

[6] https://home.a4l.org/

[7] https://www.medbiq.org/

[8] https://www.hropenstandards.org/

SCOPE

The architectural options must support:

- multiple program submissions within one state;
- more than one state collaborating on a state collection system;
- data trusts that involve working with states; and
- direct-from-employer options and third-party service options.

These infrastructure designs in the companion *JEDx Systems Architecture Technical Workgroup Report: Analysis of Architectural Standards and Pilot Options* include methods for the data collection pipeline, including:

- submitting data (including standardized data serialization and transport);
- collecting or receiving data;
- storing and securing data;
- accessing data (and controlling access); and
- extracting and aggregating data for use.

The initial use case for this API specification focuses on data collection. However, the design also could support data exchange for operational use.

Related to Scope

In addition to the architectural options, this document reflects coordination with the Data and Applications Priorities Workgroup to address other considerations such as the following:

Employer data systems architecture
The current projects will not propose standards for employer data systems architecture but will provide some indirect requirements for **interfacing** those systems to the data collection endpoints and data definition and serialization for interoperability requirements.

Accounting methods
The Data and Applications Priorities Workgroup will consider variations in accounting methods and propose a set of standard atomic data elements that support the variation of accounting rules used in the state and federal collections and for other applications of the data. This standardized set of elements will affect the data payload(s) to be supported by the infrastructure.

Data elements collected
The Data and Applications Priorities Workgroup will define the standard payloads to be supported by the various architectural options proposed.

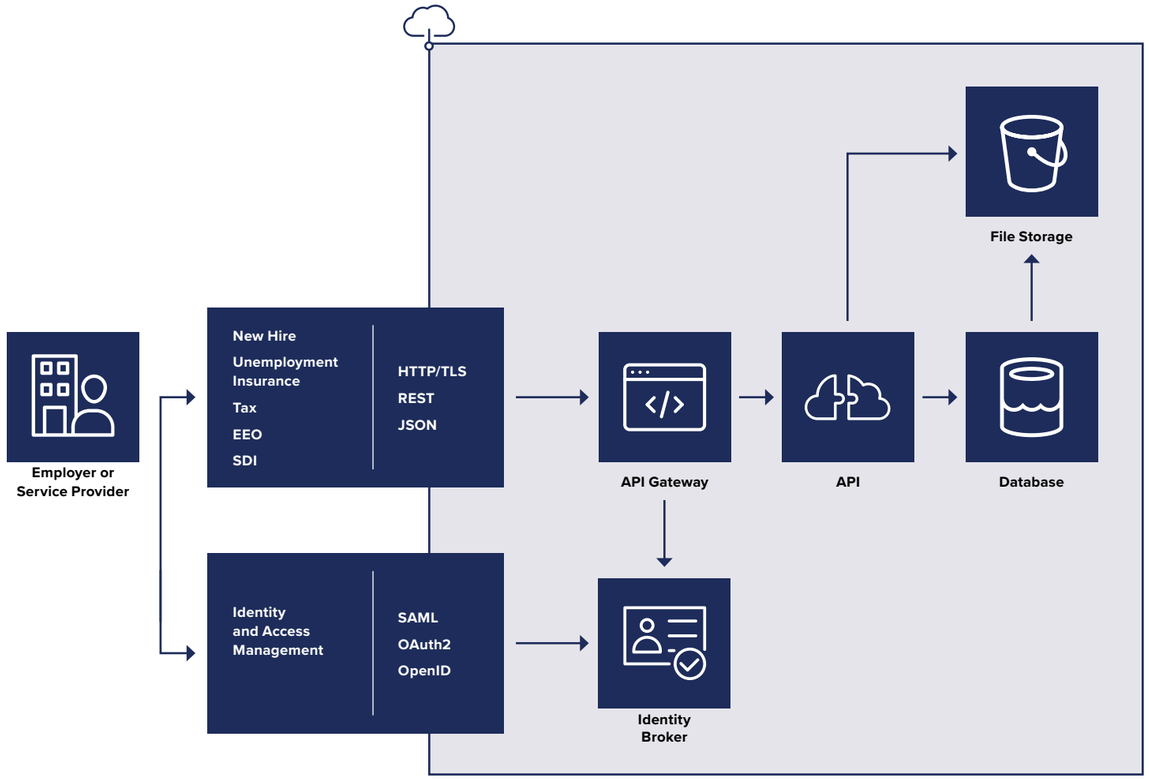
Data element definitions
The data element definitions will draw from existing data vocabulary from prior JDX+ work that is maintained by the HR Open Standards Consortium. Additional elements may be proposed for this JEDx Demonstration Project and piloting in the next phase that also will be proposed to the HR Open Standards Consortium for inclusion in the standards.

Timing of collection
The JEDx Demonstration Project determined that great potential value could be derived from collecting data more frequently using them to inform talent marketplace decision-makers (for example, employers and analysts), policymakers, and ultimately learners and workers with better insights on the most prosperous career paths and opportunities. The proposed architectural options should be capable of supporting near-real-time and event-driven collections in addition to fixed collection cycles.

Applicability to state systems
For a state that uses a data trust for intersystem and cross-sector sharing, the move to a federated hub-and-spoke architecture simplifies the physical differences between systems, provided that the hub contains a logical representation that meets the needs of all spokes (consuming systems) and that provisions are in place for abstracting the various technical/physical representations for semantic consistency. (Robert McGonough)

VISUALIZATION

FIGURE 1: HIGH-LEVEL PICTURE OF A MODERN CLOUD INFRASTRUCTURE[9]



Architectural Elements

API Gateway:
Endpoint for API. Provides security and Access Management to the API.

API: The code to execute the API logic.

Database: Structured data repository to store data collected by the API.

File Storage: Object storage to store any files, documents, or other unstructured data processed by the API.

Identity Broker: Component to provide identity and access credentials for the API.

[9] Eudy, 2022

PRINCIPLES

01

Theory of Action: The JEDx System Architecture Workgroup agreed...

- That there are multiple reasons to collect data, including business operations, decision support, economic policy, and mandated reporting in support of government functions such as tax and administration of benefits programs. Regardless, there are certain mandates, and optimizing their utility will serve all the actors in the system.
- That having complete, accurate, and timely data in context will improve policymaking and local decisions in a way that improves the lives and the success of the American workforce.
- That modern, REST-based automation will allow for the correct and timely collection of these data in a sustainable, manageable way.
- That using existing and emerging standards for both data and infrastructure is the best way to implement these goals in a long-term, sustainable structure that allows for healthy competition and cooperation and does not pick winners and losers.

02

JEDx Architecture Principles

- That automating and modernizing the data collection process of jobs and employment data between entities and the state and federal governments will increase value and timeliness.
- That using a modern, standardized technology whenever possible, but allowing for support of the current data collection infrastructure, will meet everyone's needs and move the system into a place that fulfills JEDx's vision.
- That, once the initial states test this, it will be easier for them to fully embrace it and much easier for the states that follow to upgrade.
- That the architecture design and aspects are flexible to accommodate working along varying places on the continuum between the current operational state and the fully realized JEDx vision.

03

Design Principles: How the data are moved

- That all data will be handled with privacy and security as a paramount consideration. If possible, we will use standard approaches to ensure that this is enacted.
- That using APIs will be more sustainable and future proof. That this would include:
  - Standards-based serialization
  - Possible large volumes of data—using a large worst-case example
  - Transactions in any one minute or hour of how much size per record
  - How is it going to be handled?
  - Guaranteed delivery
  - End-point creation and maintenance
  - Error management
- That allowing for binary data or nonstandardized documents to be uploaded and delivered using a file-storage and upload solution will ease the transition pains and support entities not ready to commit immediately to a fully modern system.

03

Design Principles: Data collection

- That automating how states and other aggregators collect these data internally will make the data more usable, accurate, complete, and timely.
- That unifying processes in each state will make federal collection of these data easier, and the data will be far more accurate, complete, and timely.
- That by using standards it will be possible to collect data more thoroughly and more frequently without placing an undue or additional burden on the collectors.
- That, by using standards and a standardized system and the automation that it allows, employers and providing agents will be able to both generate the data and manage year-to-year changes with less expense and with more utility for themselves.

Eudy, 2022

PRINCIPLES

Business Principles

1. Automation and data interoperability could make it easier for all stakeholders to provide and get the jobs and employment data needed.
2. More accurate data inform better policy and business decisions.
3. More timely data are more useful data.
4. Submit once, use many: Have the data be entered once in one place and then be used by the system every other time they are needed, without the user having to re-enter the data.

Technical Principles

1. Use HR Open Standards and modern, sustainable, secure techniques and methods.
2. The API infrastructure must be independent of any one data model so it can be used universally by employers, state agencies, federal agencies, and workforce organizations if required and appropriate. This approach ensures true separation of payload and transport.
3. Use the APIs in the OAGI API and Standardized Infrastructure model as a guide.
4. Make it as simple as possible so small organizations without a deep technical bench or no technical bench can use it, but make it no simpler than it needs to be. It must fulfill its purpose and be expandable into the future.
5. The API model must allow for the architecture being defined to work in a secure, privacy-aware, future-proof, and sustainable model.
6. The API model must describe how it would support the current primary use cases (see the Systems Architecture Technical Workgroup Report[10]).

[10] Eudy, 2022

USE CASES AND GOVERNANCE

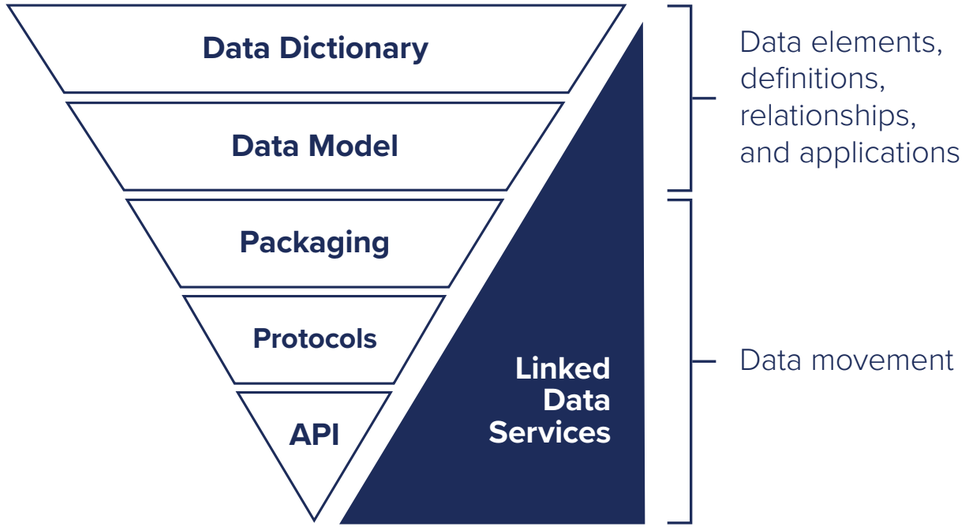
Use cases and governance patterns are defined in the Architecture Document[11] and were used to scope and frame the API design conversations.

[11] Eudy, 2022

SYSTEM MODELS

OVERVIEW

This document is focused on the data movement aspects of the JEDx vision. Data are moved against a background of data models, policy, and business needs, as well as compliance drivers. Therefore accounting for compliance, policy, and data element and object definitions is necessary to discuss the movement and collection of the data.



DATA MODELS

This document is focused on the transport infrastructure, but the context of the data being moved is what resolves the business need of the transport. The Standardized Infrastructure recommended is completely content agnostic—that is, any well-formed content can be moved over it. This section on data provides context for the next section.

The JEDx Data and Application Working Group is defining a set of data elements and objects that can be used as resources in any JEDx interactions and should be considered the authority on any JEDx data models and serialization. These data objects and the dictionary of these elements are being maintained and managed over time in the HR Open Standards (HROS) consortium.

Any of those JEDx objects can be used as a resource in the RESTful API architecture being described here. An example of a POST action using a JEDx sample resource can be seen in this document in the section labeled “Example: JEDx API Post” on page 50.

Example Datasets

UI and Workforce Collected Data Elements
Data Elements Options 04012022.xlsx

JEDx
https://www.uschamberfoundation.org/JEDx
JDX Data elements2.xlsx

HR Open Standards
https://www.hropenstandards.org/standards

Particularly useful is the 4.3 Standard, which includes the JEDx linkages and the Employee and Employment Earnings Specification.

API MODELS

OBJECTIVES

Goals

To develop and explore various architectural options for standards-based data collection and exchange that

- improve the efficiency of data collections for both employers and government;
- improve the efficiency of collections direct from employers and via service providers; and
- consider infrastructure for high-profile use cases and applications.

Target Objectives

- High-level design/architectural options for standards-based data collection and exchange to inform decisions for pilot programs
- Standards-based specifications for data transport, including an API specification suitable for pilot testing of modularized data integration, for use in pilots
- Initial guidance about possible technical and governance approaches for scalable access and use of data while protecting privacy and security
- High-level pilot-test plan with public-private cost-benefit analysis and scenario for a secure cloud-based test platform and sandbox that could be used by state systems and service providers to pilot-test with sample data

API GENERIC OVERVIEW

Overview[12]

An API is a named set of operations (functions and data) that is offered by a provider system (the reporting system) and used by consumer systems (the receiving systems of the collection) to enable communication between the provider and consumer systems (applications).

A web service (that is, a service exposed on the World Wide Web) offers a web API. A web API that conforms to the REST architectural style is a RESTful web API.

The REST architectural style was named and defined by Roy Fielding in 2000 in his doctoral dissertation that describes the web's architectural style.[13]

Fielding's description of the REST architectural style consisted of constraints in six categories:

- Client-Server
- Stateless
- Cache
- Uniform Interface
- Layered System
- Code-On-Demand

The **uniform interface** is a central feature of the REST architectural style that distinguishes it from other network-based styles. The web's components (for example, clients, servers, reverse proxy) depend on the interface uniformity for their interaction and communication.

[12] Fohn, 2019. Most of this overview is taken from the "OAGI_RESTful_Web_API_Design_EN_US" document written by Steffen M. Fohn of ADP.

[13] Fielding, Roy Thomas. Architectural Styles and the Design of Network-based Software Architectures, Doctoral dissertation, University of California, Irvine, 2000. http://www.ics.uci.edu/~fielding/pubs/dissertation/top.htm

API GENERIC OVERVIEW (CONTINUED)

Constraints

Fielding identifies four constraints of the uniform inform interface:[14,15]

Identification of Resources. Each concept (known as a resource) may be addressed by a unique identifier such as a Uniform Resource Indicator (URI). In this context, these resources could be

- a **single instance** of a resource such as a wage record, a JEDx reporting object, an HR Open employment record, or a proprietary local object for an employee or an employer; or
- a **collection of instances** such as all the wage records for a group of employers for a quarter.

These resources are represented as a URI.[17]

Manipulation of Resources through Representations.
The representation is a means to interact with the resource but is not the resource itself. Initial JEDx use cases will not take advantage of these more complex interactions.

Self-Descriptive Message.
A resource's desired state can be represented within a client's request message. A resource's current state may be represented within a server's response message. Metadata may be included to convey additional information on the resource (for example, resource state, representation format).

Hypermedia as the Engine of Application State (HATEOAS).
A resource's state representation includes links to related resources. This will not be used heavily in our initial JEDx use cases because the use case calls for pushing data to a collector rather than pulling data into an app.

RESTful Architectural Design

RESTful web APIs are designed according to the REST architectural style and leverage the existing Hypertext Transfer Protocol (HTTP) as the application communication protocol.

HTTP specifies that resources be retrieved via a unique identifier or URI that corresponds to their server-side representation. The representation of the resource may be supported by one or more formats (for example, HTML, XML, CSV, JSON, ATOM, and JPEG). HTTP serves as the basis for the uniform interface constraint of the REST architectural style (mentioned above).

Guaranteed Delivery

A modern RESTful interface ensures that the delivery of a message or a "report" is guaranteed even if the source or the target goes down. It uses a series of acknowledgments that keep the message held until the sender receives an acknowledgment that the receiver has collected the data.

Maturity Model

Leonard Richardson developed a RESTful Web API Maturity Model consisting of four levels. Each level declares an aspect of the web's Uniform Interface that a RESTful web API must satisfy for a given maturity level.[19, 20]

Level 0	• The API must use the HTTP protocol for communication transport. • At this level, HTTP is essentially used as a tunneling mechanism. For example, Web Services using SOAP (Simple Object Access Protocol) send messages with the HTTP POST method to the same URL; the operation to be invoked is communicated in the body of the SOAP message.
Level 1	• The API meets Level 0 requirements. • The API must use resources. • This level establishes resources and their management through the communication of their state representation.
Level 2	• The API meets Level 1 requirements. • The API must use HTTP verbs and HTTP response codes. • This level requires that all **Create, Read, Update, and Delete (CRUD)** data management operations performed on a resource must use the established HTTP methods (for example, POST, GET) for those operations. All message confirmations (that is, success or failure) must use the established HTTP response status codes.
Level 3	• The API meets Level 2 requirements. • The API must use hypermedia controls. • This level is referred to as Hypertext As The Engine of Application State (HATEOAS). A resource's current state representation may include hypermedia controls (that is, links) that provide the requesting system (that is, service consumer) a set of possible next steps (that is, operations) in the context of systems interacting to realize a use case.

Fielding[21] defined Level 3 as a prerequisite to being RESTful. The OAGI document this section is taken from[22] was oriented at that level of detail. However, the JEDx pilot programs will consider Level 2 as sufficient for initial JEDx use cases.

[14] Fielding, 2000
[15] Masse, Mark. REST API Design Rulebook. O'Reilly, 2011.
[16] https://en.wikipedia.org/wiki/Uniform_Resource_Identifier
[17] Gregorio, J., Fielding, R., Hadley, M., Orchard, D. "URI Template", RFC 6570, March 2012. https://tools.ietf.org/html/rfc6570
[18] Fielding, R., Gettys, J., Mogul, J., Frystyk, H., Masinter, L., Leach, P., Berners-Lee, T. "Hypertext Transfer Protocol – HTTP/1.1", RFC 2616, IETF, June 1999. http://www.w3.org/Protocols/rfc2616/rfc2616.html http://tools.ietf.org/html/rfc2616
[19] Richardson, L. "Introducing Real-World REST", QCon, San Francisco, November 2008. http://qconsf.com/sf2008/dl/qcon-sanfran-2008/slides_/LeonardRichardson.pdf
[20] Fowler, M. "Richardson Maturity Model," martin.fowler.com, March 2010. http://martinfowler.com/articles/richardsonMaturityModel.html
[21] Fielding, 2000
[22] Fohn, 2019

RESPONSE AND ERROR CODES[23]

This design will use HTTP response status codes as defined in W3C's HTTP 1.1 specification. Any modifications to the use of the HTTP response status codes in this specification are limited to changes in requirement levels (for example, change of requirement from a should to a must) or the addition of details specific to their use in a RESTful web API. The purpose of these modifications is to constrain the space of response code use to that required for partner interaction in a trading community.

There are about 60 HTTP response status codes.[24] A subset of these codes is used in this specification.

1xx **Informational**	Request received, continuing process
2xx **Success**	The request was successfully received, understood, and accepted
3xx **Redirection**	Further action is required to complete the request
4xx **Client Error**	The request contains bad syntax or cannot be fulfilled
5xx **Server Error**	The server failed to fulfill an apparently valid request

[23] Foss, 2019

[24] Internet Assigned Numbers Authority. Hypertext Transfer Protocol (HTTP) Status Code Registry. November 2012. http://www.iana.org/assignments/http-status-codes/http-status-codes.xml

SUPPORTED HTTP RESPONSE CODES

The table below summarizes the HTTP response codes supported in this specification.

Category	Code	Message	Description
2xx	200	OK	The request was successful and the server's response includes the requested data.
	201	Created	The request has been fulfilled and resulted in a new resource being created.
	202	Accepted	The request has been accepted for processing, but the processing has not been completed.
	204	No Content	The server has fulfilled the request but does not need to return an entity-body and might want to return updated metadata.
	206	Partial Content	The server has fulfilled the partial GET request.
	207	Multi-Status	The server conveys multiple status codes about multiple resources managed in the request.
3xx	301	Moved Permanently	The requested resource has been assigned a new permanent URI and any future references to this resource should use one of the returned URIs.
	303	See Other	The response to the request can be found under a different URI and should be retrieved using a GET method on that resource.
	304	Not Modified	If the client has performed a conditional GET request and access is allowed but the document has not been modified, the server should respond with this status code.
	307	Temporary Redirect	The requested resource resides temporarily under a different URI.
4xx	400	Bad Request	The request could not be understood by the server due to malformed syntax.
	401	Unauthorized	The request requires user authentication. If the request already included authorization credentials, then the 401 response indicates that authorization has been refused for those credentials.
	403	Forbidden	The server understood the request but is refusing to fulfill it.
	404	Not Found	The server has not found anything matching the request URI.
	405	Method Not Allowed	The request method is not allowed for the resource identified by the request URI.
	406	Not Acceptable	The API is not able to generate any of the client's preferred content characteristics according to the request's accept headers.
	408	Request Timeout	The client did not produce a request within a predetermined quantity of time.
	409	Conflict	The request could not be completed due to a conflict with the current state of the resource.
	410	Gone	The requested resource is no longer available at the server and no forwarding address is known.
	412	Precondition Failed	The precondition given in one or more of the request header fields evaluated to false when it was tested on the server.
	413	Request Entity Too Large	The requested resource is larger than the server is willing or able to process.
	415	Unsupported Media Type	The server is refusing to service the request because the entity of the request is in a format not supported by the requested resource for the requested method.
	416	Requested Range Not Satisfiable	The server is unable to satisfy a request for a partial resource representation expressed as a byte range in the Range header of the request.
5xx	500	Internal Server Error	The server encountered an unexpected condition that prevented it from fulfilling the request.
	501	Not Implemented	The server does not support the functionality to fulfill the request.
	503	Service Unavailable	The server is currently unable to handle the request due to a temporary overloading or maintenance of the server.

THE SERVICE ORIENTED ARCHITECTURE (SOA) ECOSYSTEM

THE SERVICE ORIENTED ARCHITECTURE (SOA) ECOSYSTEM

Overview of an SOA Architecture

A Standardized Infrastructure implementation requires three components:

1. **Authentication:** proof that an interface is allowed to interface
2. **Infrastructure:** setup to exchange information with an approved interface
3. **Data:** actual information exchanged between approved interfaces

This section focuses on the second piece—the infrastructure. For infrastructure to work, the other two components also must be in place.

Conventions[26]

The diagrams in this chapter use symbols described below. This section also explains various roles and overarching assumptions that form a lens for viewing the diagrams.

Visual Components

Application
Whether a service consumer script pulling down data or the most sophisticated service provider on the planet, this symbol represents an integrated application and its adaptor's REST API capabilities.

Connection(s)
Although a service consumer may need multiple connections, it should be directed to a bidirectional REST API from a single network endpoint, regardless of the topology being discussed.

Broker
The broker is the software that (when present) connects applications for a specific data scope by providing Standardized Infrastructure Services through REST APIs.

Operational Data Store
An Operational Data Store (ODS) readily provides and accepts Standardized Infrastructure and JEDx data through REST APIs

Roles

Infrastructure Provider
The infrastructure provider offers standardized infrastructure services for the integration.

Service Provider
The service provider is one or more applications that service JEDx data requests.

Service Consumer
The service consumer is an application that makes JEDx service requests without servicing them in return.

Assumptions

- Requests for data from service provider systems may be made by service consumers.
- Requests to create, update, and delete data in service provider systems may also be made by service consumers.
- All service consumers may participate in any one of these topologies without change, given that other systems are providing the Standardized Infrastructure and JEDx data services needed.
- Integrated components may be combined or extended over time to support multiple needs at once.
- Existing components may take on different complementary or even reduced roles to better fit with the growing capabilities of other applications and their impacts on the ecosystem.
- Diagrams represent certain ideal topologies and are used to create common points of reference.
- Components in each diagram have been arranged so that data predominately flow from left to right.

[25] Lovell, 2022

[26] Much of the content in this section is adopted from the Access For Learning Unity Adoption Guidebook available at https://cdn.ymaws.com/www.a4l.org/resource/resmgr/docs/resources/unity_adoption_guidebook_22.pdf

[27] The Standardized Infrastructure should require that all Applications participating in an integration be registered as Service Consumers, however this chapter uses a narrower definition to make a greater distinction between roles.

DIRECT SOLUTIONS

A direct peer-to-peer connection is the most common API–services connection type yet not very scalable when multiple trading partners are involved. Direct solutions grant service consumers access to the service provider without the need for a middleman. This makes this Standardized Infrastructure appropriate for the smallest of integrations, empowering handling synchronous data requests in both directions. Direct data access means that even operational activities, such as changing a single employee's wage information, could be recorded in the HR system and the wage reporting system with immediate feedback and recall.

Pros

- The simplest system possible
- No additional components necessary
- Enables simple interactions with apps
- Clear growth paths to take

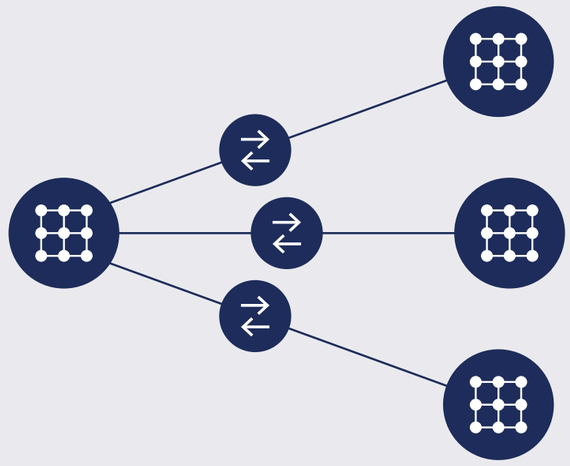
Cons

- Only one service provider per ecosystem
- Only as strong as the service provider:
  - May burden the service provider
  - May limit the service consumers
- Growth requires additional components

Roles

- Infrastructure provider: Application on left in this diagram
- Service provider: Application on left
- Service consumers: Applications on right.

In this diagram, the application on the left is the server playing the roles of both infrastructure and service provider, while the client applications on the right play the role of service consumers empowered to read and write certain data directly to the service provider.



BROKERED SOLUTIONS

At the other extreme of the data integration world is the middleware model, which is the traditional topology for Standardized Infrastructure integrations and is still the most capable. This makes the Standardized Infrastructure appropriate for large-scale integrations as the broker manages the other components, including connections and security. Brokers allow states to flow data up for reporting and back down in order to positively affect employee and wage reporting administration.

Having a brokered solution would enable the near-real-time sharing of data across internal state systems and multistate systems, ensuring objectives such as each employee's records being picked up and reported to the right place exactly when needed with no human intervention, saving money and time and avoiding human data entry issues.

Pros

- The dedicated broker manages the entire integration.
- All applications may consume and/or provide data.
- Applications can be mixed, matched, and configured to create the strongest ecosystem.

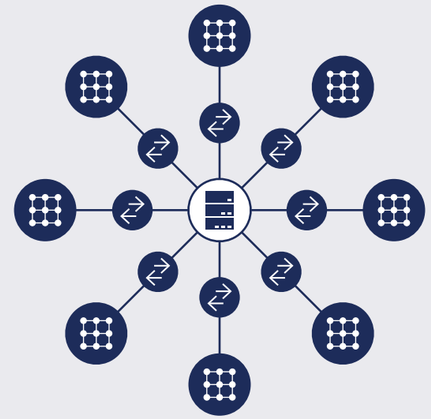
Cons

- Ensuring well-matched quality service providers for each data domain takes additional deliberate effort.
- To maintain robustness, data may need to be exchanged asynchronously.

Roles

- Standardized Infrastructure provider: Broker in the middle of the diagram
- Service provider: Some application(s)
- Service consumers: Some application(s)

This topology illustrates the advantages of using a broker. One of these advantages is maintaining the simplicity of a small integration from any one application's point of view. Instead of seven sets of certificates, credentials, services, access control lists, connections, and more, there is only one set for the broker. This model encourages more applications to participate in the integration. For example, when the state workforce agency verifies and shares the best addresses for an employer and their contacts, it doesn't just help workforce reporting, it also positively affects other systems. Systems that report on new hires now have a good address to associate with the employer. Strong systems make everyone stronger. For large data integrations, the strongest systems use this topology.



DATA HUB SOLUTIONS

For a mid-sized integration, the Standardized Infrastructure lends itself well to a data pool model, often called a data hub because it creates a place to put data so that they can be queried using the hub as the source of truth. In the workforce data collection example, a third party could act as an aggregator and service provider to the employers and states. The term “ODS” here is used generically to indicate the data storage used by the centralizing party.

The ability to control the source dataset for reporting is often desirable for ensuring data quality. This feature allows JEDx and the Standardized Infrastructure to evolve to meet a growing integration’s needs, as the ODS acts as both the infrastructure and service provider.

Pros

- No application needs to be a service provider
- One source of truth for all queries
- Queries that span multiple applications’ data can be answered

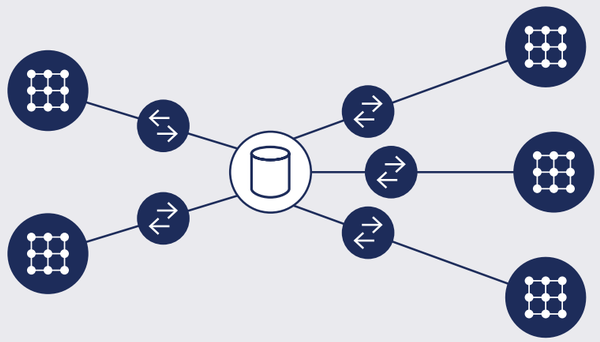
Cons

- Query results are only as current as the data sent to the ODS by its service consumers

Roles

- Standardized infrastructure provider: ODS in the middle of the diagram
- Service provider: ODS in the middle
- Service consumers: All applications

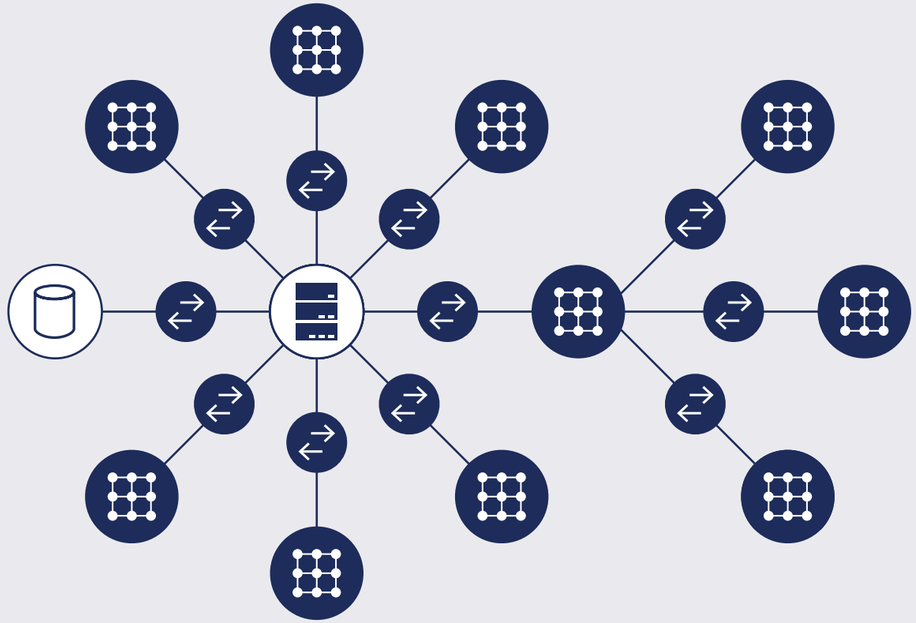
This illustration depicts the service consumers on the left supplying data through requests to the ODS in the middle, which then can handle queries from the service consumers on the right. While these are artificial limits, it helps to convey how an ODS can create a JEDx ecosystem by taking on the burdens of being the only service provider. When it comes to growth, notice that the right half of this example is the same as the direct solution described above, indicating opportunities for growth. If the two service consumers on the left side of the diagram became service providers and the ODS was replaced with a broker, the result would be smaller version of the brokered solution.



HYBRID SOLUTIONS

Having a common REST API and supporting services opens the door to many possibilities. When putting together solutions, do not be afraid to mix and match approaches to construct the best solutions possible. The following diagram shows one approach to combining all of the functionality discussed above.

Here, an ODS stores whatever data are not provided elsewhere, linked by a broker managing the integration and a service provider hosting additional service consumers. Although an integration like this one is best rolled out in phases, growing to this level of sophistication is possible. Many projects, both in the United States and in other countries, have used brokers as infrastructure providers in front of one or more service providers to create a more modular data hub.



COMPONENT: AUTHENTICATION

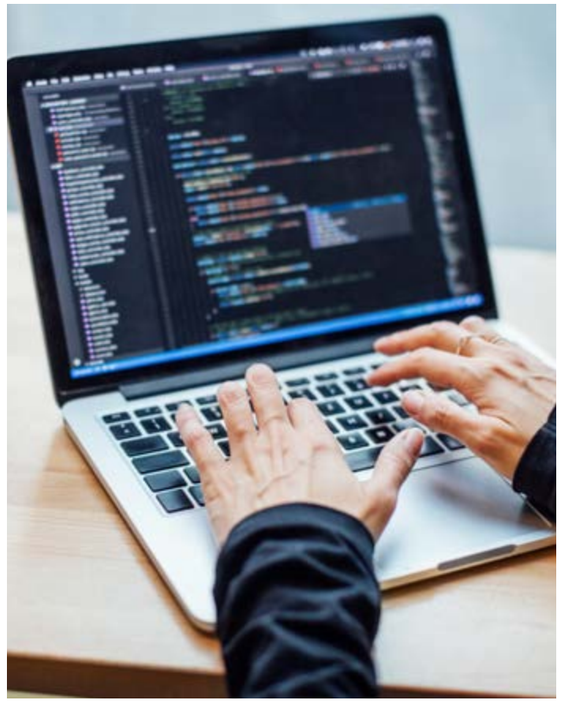
To access our JEDx objects, use **HMACSHA256** because it is the default authentication method of the Standardized Infrastructure. **HMACSHA256** has this role because it:

- starts simply with two pieces of information;
- never sends the secret over the (global) Infrastructure;
- is secured with a one-directional hash algorithm;
- identifies the service consumer in its confirmation; and
- is salted with the current date and time in UTC ISO 8601 format (contained in the timestamp element) that changes on every request to avoid replay attacks.

Additionally, using preprovisioned environments can simplify getting started.

Although the Standardized Infrastructure makes provision to use any SSO Authentication method, interoperability is possible only if the authentication systems on the client and server sides of the connection match. To maximize compatibility, the infrastructure documentation includes guidance for the following:

- Basic Authentication
- HMACSHA256
- OAuth 2.0
- Client Certificates



Getting an Authentication Token

Generating a **HMACSHA256** Authentication Token requires the following information:

- **Timestamp:** Date/Time in ISO-8601 format
- **Application Key/Session Token**: Unique ID
- **Shared Secret:** Used to hash the above information

There are four steps to creating a usable Authentication Token so to preserve the authentication method employed and the sender identity.

Operation Example Value

Concatenate the applicationKey and date/time, and separate them by a colon.

RamseyPortal:2013-06-22T23:52-07

Calculate the HMAC SHA 256 value using the unsent Client Application shared secret, and then Base64 encode.

6TVgYwbAhmQc2zALda8ZupfrzfqZ+XD7f2bMA0AzWRo=

Combine the applicationKey with this string, and separate them by a colon.

RamseyPortal:6TVgYwbAhmQc2zALda8ZupfrzfqZ+XD7f2bMA0AzWRo=)

Base64 encode the result and prefix it with the authentication method and a space.

HMACSHA256 bmV3OjZUVmdZd2JBaG1RYzJ6QUxkYThadXBmcnpmcVorWEQ3ZjJiTUEwQXpXUm89Cg==

Example JavaScript

The following code runs in Node.js and is used to generate the following examples. Replace the “PUT_SESSION_TOKEN_HERE” and “PUT_SHARED_SECRET_HERE” with your corresponding production values.

```
var CryptoJS = require("crypto-js");
var timeStamp = (new Date()).toISOString();
var sessionToken = "PUT_SESSION_TOKEN_HERE";
var sharedSecret = "PUT_SHARED_SECRET_HERE";
var valToHash = sessionToken + ":" + timeStamp;
var hash = CryptoJS.HmacSHA256(valToHash, sharedSecret).toString(CryptoJS.enc.Base64);
var authToken = "HMACSHA256 "
+ CryptoJS.enc.Base64.stringify(CryptoJS.enc.Utf8.parse((sessionToken) + ":"
+ hash));
console.log("Timestamp: " + timeStamp);
console.log("Authentication Token: " + authToken);
```

Using the Token

When you have computed your authentication token, you may use it with the Standardized Infrastructure in the HTTP authorization header. When servicing a request, it is up to the software handling the request to verify the token’s validity and access level.

Example Header

Authorization : HMACSHA256
UFVUX1NFU1NJT05fVE9LRU5fSEVSRTpYdk0zbGpxV3VJMXV2UnhZVkttUkFsV3hlTFVhM2orbEdNUUwvWk52NzdFPQ

MESSAGE PARAMETERS

The primary function of any API is to enable secure and robust exchanges of service consumer-issued requests and service provider-issued responses, including events, over a Standardized Infrastructure-conformant REST transport layer. Every message exchanged has any of several elements, provided by the sender, that specify the source of the message and the security, destination, and context information. In REST, these elements are carried over HTTP(S) and may carry any of the following values:

- A payload: Typically in XML or JSON format
- Fields in an HTTP header (The name of the HTTP parameter is case insensitive as per HTTP specification.)
- Matrix parameters in a request URL (located after the last URL path segment only. The name of the matrix parameter is case sensitive as per HTTP specification.)

  Example: **https://.../employee;zoneId=ADP (will return students of zone 'ADP')**

- URL query parameters in a request URL (after the '?'. The name of the URL query parameter is case sensitive as per HTTP specification)

  Example: **https://.../employee?changesSinceMarker=xyz**

The table below lists all defined HTTP headers, matrix parameters, and URL query parameters as defined in the API specification. Although the list is extensive, most parameters are optional and/or have a default value. Please refer to the Open API specification of each locale's API for details on which parameter can or must be used for a specific request or response.

Parameter Details Summary

The request, response, and event columns use the standard characteristics:

- **O:** Optional
- **M:** Mandatory (required)
- **C:** Conditional (see explanation column for details)

The conveyed column uses the following abbreviations:

- **H:** HTTP Header
- **Q:** URL Query Parameter
- **M:** URL Matrix Parameter

When more than one conveyance is used or a conditional is indicated, see the explanation for details of its use.

Parameter Name	Req.	Resp.	Event	Convey	Explanation
accept	O			HQ	Used to indicate when the format is expected in the response (for example, application/json). If omitted, it may also be indicated by including an extension in the URL's path (for example: ".json"). Otherwise, results will be conveyed using the default, application/xml.
accept-encoding	C		C	H	Indicate what payload encoding is accepted in the response. Valid values are identity (not compressed) or gzip (compressed).
access_token	MC			Q	The token used to authenticate the sender of the message, authorizing the requested action. It is usually the token/hash value of the authorization header. This query parameter is only required when the authorization header is not set or another authentication standard is leveraged.
applicationId	M			Q	A unique Id for an application regarding the Data Privacy Enforcer service. It is different from the applicationKey (see below) that is used for authorization.
applicationKey	MC			HQ	If the application key is not contained in the authorization header, then this header must convey this key together with the authentication. The consumer may choose to convey this value in either place, so providers must honor it in either place.
authenticatedUser	OC			H	Set to the user's identification (depending on the authentication used) when verified by the middleware. The receiving service provider can trust this field by confirming the middleware's credentials.
authenticationMethod	MC			Q	The identifier for the authentication method used. Note that placing basic access authentication information in a URL query parameter is highly insecure and should not be used in any production systems. Unless otherwise specified, the prefix from the authorization header is used: **HMACSHA256/Bearer/Basic**
authorization	MC		MC	H	Used to authenticate the consumer and is the basis for determining whether the consumer has the necessary authorization to issue the request or publish the event. When conveyed in URL Query Parameters, access_token and authenticationMethod are used.

MESSAGE PARAMETERS (CONTINUED)

Parameter Name	Req.	Resp.	Event	Convey	Explanation
changesSinceMarker	OC	OC		HQ	Request: URL Query Parameter. Only required if a change was made since the request was performed. Response: HTTP Header. Only required if the request had the changesSinceMarker as a URL query parameter and no paging is used, or if paging is used but the first page is requested.
connectionId	MC			H	Identifies the established connection over which the next message in the queue is being requested and delivered. This must be a unique unsigned integer ranging in value from 0 to one less than the current value of maxConcurrentConnections.
content-encoding	C	C	C	H	Indicates the payload encoding. Valid values are identity (not compressed) or gzip (compressed). See also: accept-encoding
content-Type	MC	M	M	HQ	Tells the receiver how to parse the body of the message. Supported generally, however, data models are generally conveyed with types application/json or application/xml (default). Must be conveyed whenever a body is present. May be omitted in a request. In that case, the mime type is either • the mime type indicated on the URL (that is, .json)l or • XML if not defined on the URL or the HTTP header. See also: Accept
contextId	O	O		HM	The "context" of the service provided. The range of possible context token values for a given object or functional type service is defined by either or both the data model that the environment is supporting and the administrators of the zone. If not provided, it will default to DEFAULT. This is carried as a matrix URL parameter in request but is conveyed as an HTTP header field on events. See also: relativeServicePath
contractId	O			Q	Used in conjunction with the Data Privacy Enforcer Service. It allows specifying a contract that is in place between two parties and what exact filer rules apply.
deleteMessageId	OC			M	The ID of the last message received and processed by the queue owner. It is used only when making query requests to the queue service instance when deleting a previously retrieved message from the queue.

Parameter Name	Req.	Resp.	Event	Convey	Explanation
environmentURI		OC		H	May be returned by the environment provider where the environment is preprovisioned.
etag	OC	O		H	Optionally returned by a service provider within a query response, equivalent to a "checksum" on all the objects of the type being queried, which are maintained by the service provider. If it is returned in a response, the consumer may include it the next time it issues any query to that service provider, so that the service provider can determine whether it needs to respond to the query at all. (If the ETag value submitted matches the current value, no data has changed, so the response will be no different from last time.)
eventAction			M	H	The specific type of event being reported: CREATE, UPDATE, or DELETE
fingerprint	MC		MC	H	Unique environment identifier that can be safely shared with others. In order not to compromise security, it must not match the environment's refId, sessionToken, userToken, or applicationKey. Added by the broker to all requests before forwarding to the service provider. Added by the functional service provider to events that are intended only for the job's owner.
generatorId	O		O	H	The optional identification token of the "generator" of this request or event (for example, the administrative clerk who entered in the data that was responsible for generating a create request).
messageId	O	M	M	H	UUID that uniquely identifies the message that carries it.
messageType	O	M	M	H	One of: EVENT, REQUEST, RESPONSE, or ERROR If not provided, it will default to REQUEST.
methodOverride	MC			H	HTTP PUT: Included in an HTTP PUT message when it is conveying a multiple-delete request because an HTTP DELETE is not allowed to have a payload. Valid values are DELETE or UPDATE. HTTP POST: Included in an HTTP POST message when it is conveying a QBE request because the HTTP GET is not allowed to have a payload. Valid values are GET (QBE) or POST (Create).

MESSAGE PARAMETERS (CONTINUED)

Parameter Name	Req.	Resp.	Event	Convey	Explanation
mustUseAdvisory	O			HQ	Informs the service provider that if the "suggested" RefId in the request cannot be assigned to the new object, the request should be rejected. Valid values are TRUE and FALSE. Used in create requests to object services.
navigationCount		O		H	The total number of objects in the set of results generated by the initial paged query that is associated with the returned navigationId.
navigationId	MC	O		H	Identifies state maintained in the service provider for the consumer issuing the paged query request. If returned, the consumer must supply the navigationId value when requesting subsequent pages of that object type from that service provider. This should not happen when queryIntention is set to NO-CACHING or ONE-OFF.
navigationLastPage		O		H	Included as an aid for the consumer in detecting when to stop issuing paged query requests.
navigationPage	O	MC		HQ	The number of the page to be returned. If it is outside the range of results (which does not constitute an error), an HTTP response with a code of 204 (No Content) will be returned. The first page is indicated with the value 1 (that is, navigationPage=1).
navigationPageSize	MC	MC		HQ	Included in every paged query request and indicates the number of objects to be returned in the corresponding response page. If the page size specified is too large for the service or environments provider to supply, an error with code 413 (Response Too Large) will be returned. When contained in the response, it indicates the actual number of objects on the returned page.
Order	O			Q	Orders the result set by one or more specified elements and directions. For example, [name/nameOfRecord/familyName=ascending;name/nameOfRecord/givenName=descending]. See discussion in Section 5.8.
partyId	M			Q	A unique ID for a party regarding the Data Privacy Enforcer service. For example, a party can be an organization, employee, aggregator, or jurisdiction.
podId	O			H	The unique ID (UUID) of the POD that was applied to a payload. The Data Privacy Enforcer service may set this HTTP header.

Parameter Name	Req.	Resp.	Event	Convey	Explanation
podVersion	OC			H	The POD version of the POD that was applied to a payload. The Data Privacy Enforcer service may set this HTTP header. If this HTTP header is provided, it is expected that the podId HTTP header is also set.
queryIntention	O			H	If the consumer intends to follow up with further paged queries after this one, this field must be included in the paged query request. Valid values are: ALL, ONE-OFF, or NO-CACHING ALL: The consumer intends to come back for the remaining pages of data. It is expected that the provider would return a navigationId HTTP header in this case. ONE-OFF: The consumer intends to make only this query, however the results may come from a cached source. NO-CACHING: The consumer needs the data returned as they currently exist in the provider's data store. It is a hint to the service provider that maintaining consumer state (and supplying a navigationId) would be advantageous. When not provided the default value is ONE-OFF.
queueId	MC	C		H	Contains the identity of one of the consumer's assigned queues to which the delayed response or job object events from the service provider related to this request must be routed. If a service provider is in true delayed mode, then the consumer's queueuId must be provided in each response to the response connector. See also: requestType, jobId
role	O			Q	A role name regarding the Data Privacy Enforcer service. It is expected that it be used in conjunction with the applicationId (see above) to further tie down privacy rules to a particular application role (for example, teacher or student). Depending on the role, different privacy rules may apply.
relativeServicePath		MC		H	Replicates all information contained in the segments of the request URL following the request connector, potentiallyincluding the service name, extended query template name, or service path defining the payload format, and any accompanying URL matrix parameters (context and zone). URL query parameters are included. The environment provider places it into all delayed responses (and would therefore not be supplied by a service provider in a brokered solution), as an aid to stateless consumers. It is optional for immediate responses.

MESSAGE PARAMETERS (CONTINUED)

Parameter Name	Req.	Resp.	Event	Convey	Explanation
replacement			O	H	Set to FULL (current values of all object elements) or PARTIAL (only elements whose values have changed) If not set, it is defaulted to PARTIAL.
requestId	MC	MC		H	Only required for delayed requests. A consumer specified token that uniquely identifies every delayed request issued by the consumer. It could be as simple as a monotonically increasing integer. It is uUsed to correlate the delayed (asynchronous) response with the original request.
requestAction	O			H	Indicates what the request is trying to do. Defaults: • POST: CREATE • PUT: UPDATE • DELETE: DELETE • GET: QUERY • HEAD: HEAD
requestType	O			H	One of IMMEDIATE or DELAYED. If not set, it defaults to IMMEDIATE.
responseAction		M		H	This must exactly match the requestAction value contained in the HTTP header of the request being responded to. Valid values are CREATE, UPDATE, DELETE, QUERY, or HEAD.
serviceName			M	H	The name of the data object collection being conveyed in the event.
serviceType	O	O	O	H	One of UTILITY, OBJECT, FUNCTIONAL, SERVICEPATH, XQUERYTEMPLATE, or SERVICE. If not provided, it will default to OBJECT.
serviceSubType	O	O	O	H	If a service is for an admin directive rather than the actual base service, then this HTTP header must be set to “adminDirective.” If not provided, it must be assumed that the request, response, or event is intended for the base service.
sourceName	MC			H	The applicationKey is added by brokered solution to all requests before forwarding to the service provider. Used by the service provider in brokered solutions when issuing an alert concerning an erroneous request.

Parameter Name	Req.	Resp.	Event	Convey	Explanation
timestamp	MC	M	M	HQ	Date/time of event creation (in ISO-8601 format, which is also used as the basis of xs:dateTime) If there is no need for authentication, it may be omitted from the request. If needed, this value may be provided as a URL query parameter instead of a header.
vary		O		H	This HTTP header can be set by the provider to indicate that it supports compression. It would be set only if the consumer or broker calls the provider with uncompressed payloads where the provider could deal with compression. In such a case, the HTTP header vary would take the value Accept-Encoding
Where	O			Q	A restricted XPATH expression that identifies which among the set of all objects supplied by the provider will satisfy the query and be returned.
zoneId	MC		M	HM	Indicates the zone to which the request should be routed. It is a token that must have a value that • is unique from any other zone ID; and • identifies an entry in the zone registry if that registry service is present. If not specified in the consumer’s request, the request connector will insert the consumer’s “default” zone ID (assigned to the consumer when it initially created its environment). This is normally carried as a matrix URL parameter in requests, but it is conveyed as an HTTP header field on events.
[name matches eXtended Query Template parameter]	MC			Q	Supplies value to the parameter

GETTING STARTED QUICKLY

GETTING STARTED QUICKLY

Part of the challenge of implementing the JEDx API is to make end-point creation and maintenance as simple as possible to support both large employers and small ones with limited resources. The Access For Learning (A4L) open source community maintains several framework Software Development Kits (SDKs) on GitHub that support the Standardized Infrastructure.

GitHub Repository	Language	Version	Short Description
DSU Java Framework	Java	Infrastructure 3.x	SDK for developing 3.x Adapters for Java. This is an “infrastructure” SDK, and therefore the framework can be used with any locale data model (JEDx, HR Open Standards, UI Reporting Objects) independent of content.
DSU .NET Framework	C#	Infrastructure 3.x	SDK for developing 3.x Adapters for C# (.Net). This is an “infrastructure” SDK, and therefore the framework can be used with any locale data model (JEDx, HR Open Standards, UI Reporting Objects) independent of content.

Sandbox Endpoints

The sandbox will have the following features:

- POST. PUT. GET, DELETE endpoints
  - Against the JEDx objects defined
- SAML Authentication actions
  - Registry of approved employers, aggregators, and vendors allowed to act on the endpoints
- A list of resources and groups of resources

Other Implementation Resources

Implementation resources can be found at the following sources:

- Architecture Confluence Wiki
  https://a4ldocumentation.atlassian.net/wiki/spaces/ARCHITECTU/overview
- SIF 3 Infrastructure API Documentation (3.4)
  http://specification.sifassociation.org/Implementation/Infrastructure/3.4/ServiceDocs/
  - These documents have the technical information necessary to implement the API.

EXAMPLE: JEDx API POST

EXAMPLE: JEDx API POST

This section demonstrates how a POST action of a collection of employee wage records would look. It eventually will reflect the objects that come out of the JEDx Data and Applications Workgroup.

Example: POST of an Employee Wage Record in JSON

```
POST /hr/v1/JedxEmployees HTTP/1.1
Host: alpha.com
Accept: application/json
Content-Type: application/json
Authorization: HMACSHA256 bmV3OjZUVmdZd2JBaG1RYzJ6QUxkYThadXBmcnpmcVorWEQ3ZjJiTUEwQXpXUm89Cg==
Timestamp: 2013-06-22T23:52-07

{
 "JedxEmployees": {
            "JedxEmployee": {
            "EmployeePersonalRefId": "1652D3E34F419D75101A8C3D00AA001A",
            "EmployerID": "9998887777",
            "Job": {
            "SOC": "003",
            "title": "Manager"
            },
            "LocalId": "946379881",
            "Name": {
            "FirstName": "Charles",
            "LastName": "Woodall",
            "MiddleName": "William",
            "Prefix": "Mr.",
            "Type": "04"
            },
            "OtherIdList": {
            "OtherId": [
            {
            "Type": "0004",
            "value": "333333333"
            }
            ]
            },
            "RefId": "D3E34F419D75101A8C3D00AA001A1652",
            "SSN": "9998887777",
            "StateProvinceId": "C2345681",
            "Wage Record": {
            "quarter": "4",
            "value": "9546.45"
            }
```

PRIVACY

JEDx PRIVACY PROTOCOL (JPP)

We are suggesting that the JEDx Privacy Protocol (JPP) be based on he Global Education Privacy Standard (GEPS),[28] which is a prekindergarten through workforce (PK-20) global set of data privacy obligations data structures that can be aligned to contractual clauses as well as technical control benchmarks. JPP will include open XML (or JSON) code (PODS) to transfer privacy obligations between controllers and processors, bridging the gap in understanding data protection expectations. JPP allows organizations to choose the JPP standard suggestions or use other existing standards (for example, IEEE, NIST, or ISO) to set expectations between vendors and customers for managing employee data. This standard was developed for student data as well as workforce and employer data.

We suggest that end users and marketplace providers get involved proactively in this work to assure that their needs are met. As the Alliance grows and demands for better data privacy oversight increase, this work will be the foundation for how products validate their controls over employee data.

Privacy Obligation Document (POD) Components

1. **Privacy Obligation Document (POD):** An artifact derived from a paper contract that contains details of the parties involved, the data that can be transferred from one party to another, details of the technical benchmarks that must be adhered to (for example, encryption levels), and details of any additional parties that may handle the data.
2. **Privacy Obligations Registry Utility (POD Lookup) Service:** This service provides a means for external applications to request and obtain the current POD that applies to them.
3. **POD Enforcer:** Officially the "Data Protection Enforcer Service," this service
   - checks that any incoming requests from external applications are referencing their correct POD;
   - uses the rules from the applicable POD to clean the raw data being returned in a request, ensuring that a "cleansed" dataset is returned to the requesting external application; and
   - is placed and configured to honor all payload encryption requirements.

Privacy for the Service Provider

A signed standard contract (paper or otherwise) forms the agreement between the data provider (typically the employer or an aggregator for multiple employers)—known as the data controller—and the downstream consumer of the data known as the data processor.

A POD is created by breaking down the contract into its various clauses, linking these clauses to obligations, and finally defining benchmarks that call out the standards a data processor is expected to honor when they handle the data. Once a POD has been created, it can be enforced in any environment. In Standardized Infrastructure/JEDx-enabled environments, the POD lookup service and the POD enforcer are needed to ensure that the right POD is referenced and that the filtering is being applied. In the simple example below, a data processor (for example, a third-party attendance package) is assumed to have already obtained the correct POD and makes a request of the data controller (for example, an employer).

1. The POD is registered with the POD Lookup Service, with its identifying Data Privacy Marker (DPM) using an HTTP PUT request to the Privacy Obligations Registry Service.
2. The POD is made available to the POD enforcer with its identifyingDPM.
3. Data requests from data processors include the applicable DPM and are mediated by the POD enforcer.
4. For each data request received from a data processor, POD enforcer
   - ensures that the POD identified by the DPM provided in the data request is the most up-to-date and current POD for that data processor; and
   - applies the data cleansing rules specified in the applicable POD to the data received from the service provider before the cleansed dataset is returned to the data processor.

Privacy for a Data Processor

A data processor (that is, an employer service provider aggregating multiple employers) is assumed to have already obtained the correct POD and its identifying DPM and makes a request of the data controller (that is, thecollecting agency).

1. A data request is made, including a DPM that identifies the applicable POD.
2. The POD is checked by calling the POD lookup service using the DPM.
3. The data request is forwarded to appropriate backend systems and raw data are extracted. The POD enforcer applies the data-cleansing rules from the applicable POD, ensuring that the data conform to the fields and other conditions the data processor is entitled to receive.
4. Filtered data are returned to the data processor.

APPENDICES

APPENDICES

Appendix A: Example API Models

SIDES**:** https://www.naswa.org/services/sides

ADP OAGI: SCORE Open Source API Model: https://oagiscore.org

GitHub for SCORE Product: https://github.com/OAGi/Score

OAGi White Papers & Specs: https://oagi.org/Resources/TechnicalResources/WhitePapersSpecs/tabid/197/Default.aspx

ADP Presentation on Enterprise API Management at NIST Open Industrial Digital Ecosystem Summit: https://oagi.org/Portals/0/Downloads/Meetings/2019%20NIST%20Summit/adp-nist-oagi-summit-2019-06-04.pdf

SCORE Presentation/Demo by Scott Nieman from Land O'Lakes (an OAGi member): https://s3.amazonaws.com/aggateway_public/AgGateway+Education+%26+Action+Week+Recordings/SCORE+Overview+.mp4

DSU Standardized Infrastructure: DSU Standardized Infrastructure

Appendix B: Background: Development of PODS in K12

Schools, administration officials, classroom teachers, and education authorities are faced with an overwhelming array of applications to assist in a student's learning, school administrative functions, and community communication tools.

Data may travel from on-premises, highly trusted sources to external applications that may not be as trusted. A paper or digital contract is the only primitive protection for this data exchange.

Access For Learning's (A4L) SIF Infrastructure Specification 3.3 introduces an approach that specifies a machine-readable form of the contract and provides a mechanism that governs the process by which an external application requests data. By referencing the correct version of the contract in each request, appropriate filters can be applied to the data returned on each request, thereby ensuring that the external application receives only the information that it should. By referencing the digital contract, the external application also acknowledges certain obligations that it must adhere to when handling the returned data. This digital contract is called a Privacy Obligation Document, or POD.

JEDx

U.S. Chamber of Commerce
Foundation